FORTNITE ON ANDROID LAUNCH TECHNICAL BLOG

9.6.2018

By The Fortnite Team

EXHIBIT 10004

INTRO

On August 9th we launched the Android Beta of Fortnite on select devices from our partners at Samsung. A few days later, we began issuing invitations to a subset of Android device owners across a variety of handset manufacturers. We've learned a lot from the beta with regards to performance, security, device compatibility and providing Fortnite on Android through the Fortnite Installer. Let's dive into the technical details behind the launch, what we're working on and where we see Android in the near future.

STATS

In the first 21 days since the Fortnite's launch on Android, interest has been extremely high, with over 23 million players entering our Android beta and over 15 million players installing our APK. While we are in an invite-only phase for Android, our conversion from players being invited to playing is similar to that of the iOS beta.

HARDWARE, OPTIMIZATION

TECH INTRODUCTION

Shipping the same game across all platforms while supporting cross-play presented a unique challenge. Usually, when trying to scale a game down for mobile devices, you simplify the content and even design, in order to fit within the performance constraints of the platform. For instance, you might cull objects closer to the camera to reduce draw calls. In Fortnite, Android players can be in the same match with their friends on PC and console, so we must render everything that affects gameplay.

THE PATH TO SHIPPING ON ANDROID

Since January 2018 we have been hard at work with a significant team on the Android version of FNBR. While much of our work to make this possible was spent on rendering performance, stability and memory, the sheer number and variety of Android hardware, OS versions, and driver versions was the major hurdle we had to overcome.

Working with partners has been crucial to bringing Fortnite to Android. Without their knowledge, expertise, and hard work it would not have been possible.

We worked closely with Samsung to profile and optimize Fortnite for their devices. Samsung sent engineers to multiple Epic offices worldwide and worked directly with our engineers on optimizations and performance analysis, and contributed many code changes especially for the Vulkan renderer. Using instrumented test phones and their in-house engineering tools, they were able to give us insight into what our performance and memory bottlenecks were and how to address them. We also worked with Samsung to create the most seamless and secure installation process possible for users of Samsung phones.

Google engineers also visited us on-site to profile and optimize Fortnite, helping us to identify key optimizations, a memory leak, and also to work out a solid frame pacing implementation for OpenGL on Android. The Android engineers at Google are very talented and passionate about making the Android ecosystem awesome for gaming and constantly improving it.

In addition, we've been able to work with numerous other partners to test and optimize Fortnite including ARM, Qualcomm, Imagination Technologies, Razer, HiSilicon, and many

others.

FRAGMENTATION

The Android ecosystem consists of phones made by several different manufacturers. Each phone is designed around a System-on-Chip (SOC) which includes a configuration of CPU and GPU cores. There are several common families of SOCs such as Snapdragon by Qualcomm (71% of currently supported devices) which contains an Adreno GPU, and Exynos by Samsung, the MT series by MediaTek and the Kirin series by HiSilicon all of which contain ARM Mali GPUs. Each device ships with a slightly different version of the Android operating system and most manufacturers customize the scheduler and power management features. Devices that have the same GPU also ship with different graphics driver versions. The net result is that two devices that share the same underlying hardware can have very different performance characteristics and are also subject to different bugs.

We were pleasantly surprised to see that adoption of the latest version of Android is actually much stronger than we anticipated for 0-2y old devices. Currently in our Beta, with higher device requirements, we see more than 92% of Fortnite users are running Android 8 (Oreo) or newer, ~8% are on Android 7 (Nougat), and the less than 0.5% are on a version of Android released in 2015 or earlier. For 2y and older devices, there's a significant drop-off. For those devices, manufacturers need to release customized updates and in most cases they must coordinate with the various wireless carriers around the world to develop and release these updates.

To manage this complexity, we use the hierarchical device profile system in Unreal Engine. We start by creating four performance profiles: Low, Mid, High, and Epic. These profiles adjust scalability settings in the engine to allow the game to run on devices with different performance characteristics. Low pulls in view distances as far as possible and disables all optional graphics features. Epic has everything turned on: shadows, foliage, and the farthest view distance that can run on the latest devices.

On top of this, we have a set of GPU profiles, e.g. Adreno 54x and Mali G72. These GPU profiles choose a performance profile that best fits the hardware's capabilities as well as allowing us to enable optimizations or workarounds needed for that specific hardware. Finally we have a set of device-specific profiles, e.g. Samsung Galaxy Note 9 Adreno and Google Pixel 2 XL. This final layer of device profiles allow us to enable further workarounds or optimizations on specific devices where needed. At startup we look at properties such as the device model, OS version, and graphics driver version to determine which device profile is applied.

RENDERING PERFORMANCE

The CPU cost of rendering was by far our biggest performance bottleneck to maintaining a solid framerate. We spend much more CPU time in the graphics driver on Android compared to PC, console and iOS. While the most recent Android devices such as the Adreno version of the

Galaxy S9 can handle more than 1500 draw calls per frame, older devices can handle far less. Midrange devices that we support need to average around 600 draw calls while the lower end of devices need to average closer to 400.

We had already reduced the number of objects we need to render each frame about as far as is practical when we shipped on iOS earlier this year so we had to pursue other options. We experimented with dynamically batching draw calls into instanced draw calls and while it did provide some gains they were not substantial enough to be worth the added code complexity.

A number of optimizations in our OpenGL renderer gave small wins, but one of our biggest wins was a surprise and came as part of one of our memory optimizations: emulated uniform buffers. This is a code path UE4 has supported for years and is used for OpenGL ES2 devices that do not have native support for uniform buffers, aka constant buffers. Here's how it works. At shader compilation time we identify all constants needed for a shader and pack them into an array from which the shader reads. We also store a mapping table that tells the engine where to gather constants from uniform buffers and where to place them in the constant array. At runtime, we keep uniform buffers in CPU accessible memory, use the mapping table to copy them to a temporary buffer, and upload all constants with a single glUniform4fv function call.

While it's something of a mystery, the emulated uniform buffer code path saved us a substantial amount of time in the driver. It's possible that the driver is doing something similar internally and our implementation is faster for our workloads. It's also possible that since we are creating and binding fewer resources the driver simply spends less time on bookkeeping for lifetime management of those buffers. Whatever the case, this path reduced driver cost by 10-15% in some cases.

VULKAN

Vulkan is the newest graphics API for Android and designed for performance and low-level access to the hardware. In 2016 we partnered with Samsung to create the ProtoStar demo showcasing what was possible with a high-performance graphics API on Android. Since Fortnite requires us to draw so many objects each frame, Vulkan seems a natural fit. Unfortunately, it wasn't so clear-cut.

Vulkan support is not yet a requirement on Android so we cannot yet count on it being available across the entire ecosystem. Since we are only targeting more recent hardware that wasn't a showstopper for us but we ran into several bugs and performance problems on early Vulkan drivers, the net result being that OpenGL is faster and more stable than Vulkan on most devices. That's not surprising: the industry have had a decade to optimize and harden their implementations of OpenGL. Vulkan is a more complex API and will take some time to reach the same level of maturity.

We did ship support for Vulkan on the Samsung S9+ (Adreno) and Note 9 (Adreno). Working closely with engineers at Samsung we were able to optimize UE4's Vulkan support to be faster

than OpenGL by 20% on average. Going forward we are continuing to work with our hardware partners to bring high-performance Vulkan support to other devices, both giving Fortnite players on Android a better experience and improving the performance of Vulkan drivers for all developers.

MEMORY

Memory is an ongoing struggle. Not only are we shipping a full console game on a mobile device but we are continuously adding new content to the game. As such we are constantly pursuing memory optimizations. Android presents some unique challenges compared to other platforms.

There is no single memory budget we can target. Each device has a different amount of memory, will have different types of applications running in the background, and potentially different policies for deciding when to evict an application for using too much memory. There is no API for querying what this memory budget is. As a data point, during one memory test we entered a match and began allocating memory until the OS kills the process. Using freshly restarted phones without launching any other applications, on a Samsung Galaxy S8 (Mali) we were able to allocate 3GB of its 4GB total memory before being terminated. On a Google Pixel 2 we could only allocate 1.8GB of its 3.6GB total memory.

The system, especially the graphics driver, allocates a lot of memory on our behalf. UE4 tracks all memory that we request from the OS directly, including an estimation of GPU memory allocations, but we don't have great visibility into what the driver is allocating. Through experimentation we discovered that much of the memory the driver was allocating in Fortnite came down to shaders.

The emulated uniform buffer optimization mentioned earlier was pursued for memory reasons. OpenGL requires that the application can query the location of any member of a uniform buffer. For instance, one can ask OpenGL the offset for the LocalToWorld matrix within the Primitive uniform buffer. In order to support this, the driver needs to keep some metadata, e.g. an offset table, in memory that includes strings for the name of each member and it needs to do so for every program.

In Fortnite we have a large variety of shaders. During a typical play session the engine will use about 2,000 different shader programs. Inside each of these shader programs the driver seems to be storing this metadata adding up to a lot of memory. We saved over 400MB of memory in Fortnite by using emulated uniform buffers!

When we first shipped Fortnite on Android, our internal testing indicated that we were fitting within the memory constraints of our target devices. We ran tests where we turned on navigation in Google maps, streamed music, and made sure we could play Fortnite without any problems. Yet once we launched we found that many players were either crashing or experiencing poor performance due to running out of memory.

When an Android phone is running low on memory, it will try to free up resources by closing applications not in use. However, it turns out that there are a number of poor behaving background applications and services out there that simply restart when the OS closes them. This actually makes the situation worse! Android closed the application to regain memory but it restarts and begins consuming just as much memory as before. Even worse, starting and stopping applications consumes CPU time so not only have we not freed up any memory, we are using a lot of unnecessary CPU resources.

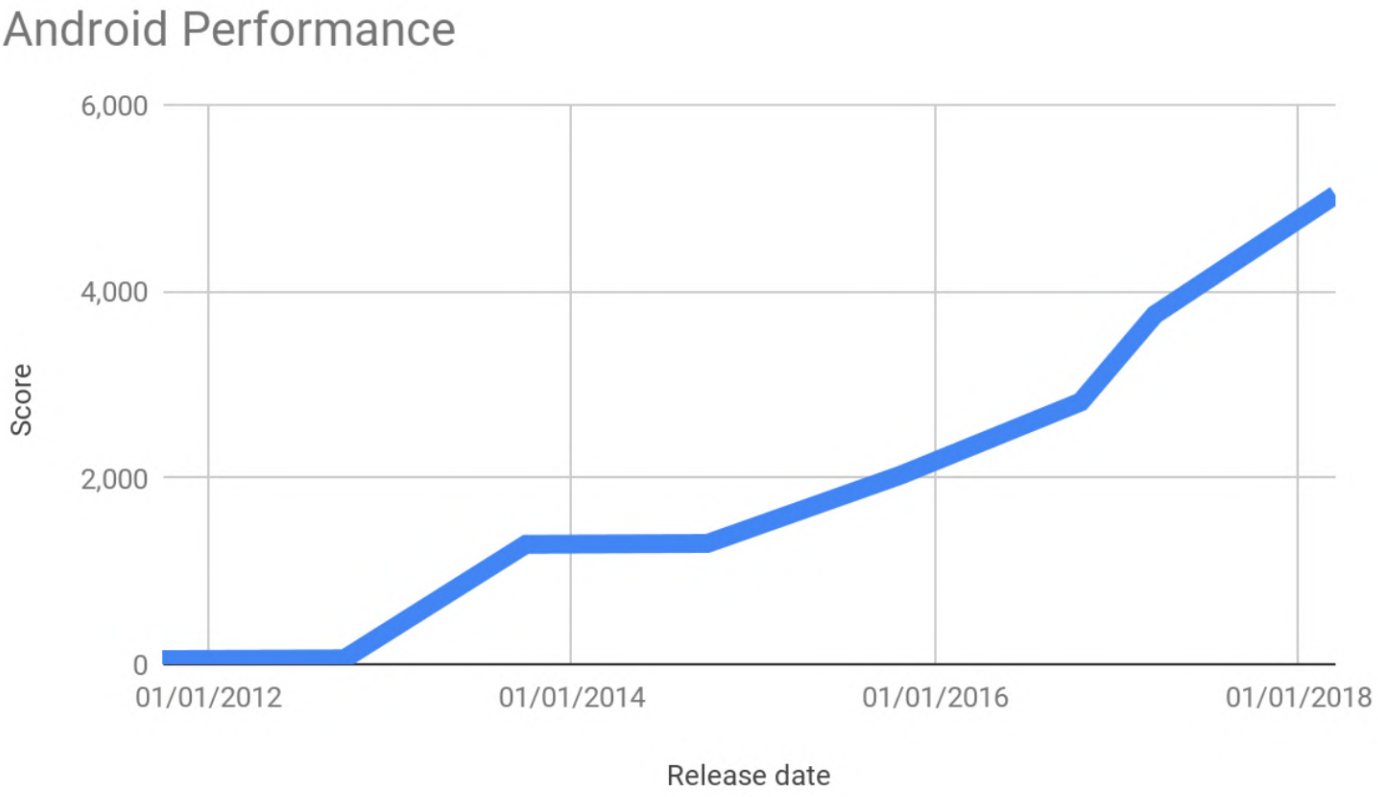
We've updated our testing processes to install and run more of the common applications that most users run so that we can find these problems earlier but we still needed to reduce memory usage and fast. We were able to narrow the problem down to users running High or Epic settings on recent Qualcomm Snapdragon devices with 4GB of memory. Again, shader memory appeared to be the main difference. As a stop-gap measure, disabling shadows helped to alleviate the problem. Doing so reduced the number of shaders required as we no longer need the variants to lookup and filter shadowmaps. We have also added an LRU cache for shaders. The cache allows us to keep only the shader programs needed to render the current scene in memory. Other shader programs are stored compressed in memory. When needed, we decompress them and hand them back to the driver.

FUTURE - NEXT STEPS

Right now we are focused on making the game run well on all currently supported devices and clawing back enough memory so we can improve both visual quality and stability. Then, we will look at compatibility issues such as sound where in the 5.40 release we fixed sound issues on many devices and opened up to support more devices. We also plan to give users more detailed control on tuning their devices and make tradeoffs between quality and performance.

We will continue to improve Vulkan support and roll it out to more devices over time. This means continuing to work with device manufacturers on optimizations, both in UE4 and in their drivers. Long term, investing in improving Vulkan support and drivers will help us provide better performance for players and all UE4 games released on Android.

We'll still be exploring older and slower phones but it's impractical to go too far back. Every year, top-end phones are getting 50% faster than the year before. Fortnite runs acceptably on two-year old phones, well on one-year old phones, and great on phones released this year. At this pace, just imagine what Fortnite will look like on phones released next year!

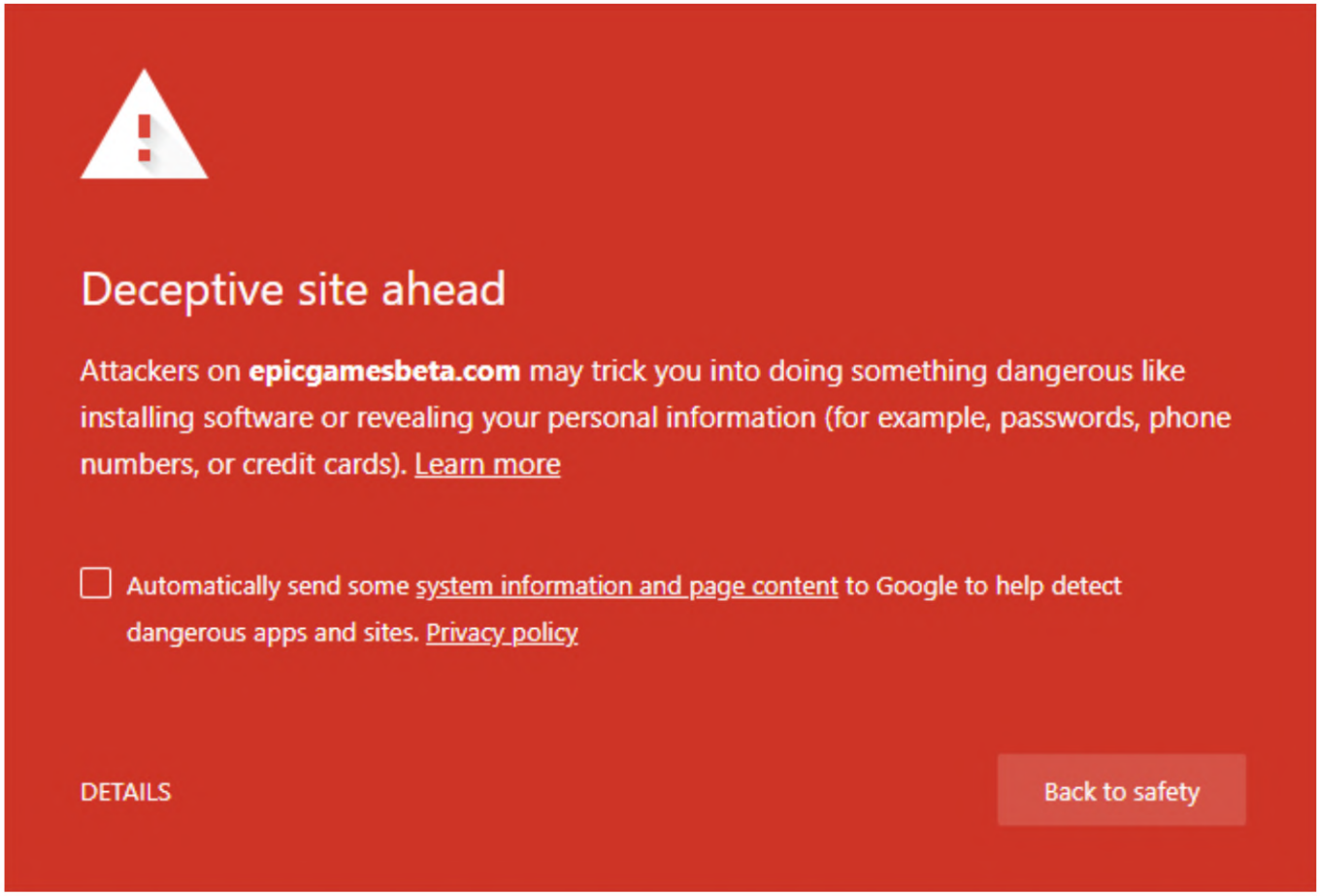


Finally, many of the code optimizations we made to the Engine have already shipped with UE4 4.20. The remaining optimizations will ship in the upcoming 4.21 release and we will continue to share future optimizations with all UE4 developers.

THE BATTLE AGAINST MALWARE

Even before we announced that Fortnite would be released on Android, we became aware of unauthorized "Fortnite for Android" websites appearing. These typically host malware or scams. As these sites come to our attention, we pursue takedown efforts with the sites themselves, the relevant hosting providers, and any ads or videos that promote them. Whether or not the website is distributing malware, we consider all distributions and alleged distributions of Fortnite for Android to be unauthorized. The only legitimate source for Fortnite for Android is directly from Epic through the official Fortnite Installer.

So far, Epic has instigated action on 47 unauthorized “Fortnite for Android” websites, many of which appear to be run by the same bad actors. We continue to police the situation with a goal of taking them offline, or restricting access by leveraging Epic’s connection to a network of anti-fraud partners (including ISPs, browser companies, and anti-virus companies) who can implement an in-browser alert like the following:



We proactively search for new malware sites as they pop up with an internal team dedicated to this task. In addition, we have also hired a third party IP and anti-fraud enforcement agency to expand our policing efforts. This partnership allows us to detect and monitor new domains that are registered with suspicious URLs so that if they evolve into malicious sites, Epic can take appropriate action up to and including litigation.

CONCLUSION

Fortnite for Android represents many industry firsts - the first console and PC game to ship on Android with full cross-play and compatibility and the first blockbuster game to ship outside of the Google Play store. It was an immense undertaking and learning process, but the rapid adoption by over 15 million Android users shows that this approach is sound and can be very successful. And most importantly all of the technical work we've done for Fortnite on Android is coming to all Unreal Engine developers with 4.21, so everyone can benefit from this work.

‹ PREVIOUS ARTICLE
(/web/20180901231727/https://www.epicgames.com/fortnite/en-US/patch-notes/v5-40-blog/)

NEXT ARTICLE

(/web/20180907123127/https://www.epicgames.com/fortnite/en-US/blog/high-stakes)

(https://web.archive.org/web/20180907123127/https://www.facebook.com/FortniteGame)

(https://web.archive.org/web/20180907123127/https://twitter.com/FortniteGame)

(https://web.archive.org/web/20180907123127/https://www.twitch.tv/fortnitegame)

(https://web.archive.org/web/20180907123127/https://www.youtube.com/epicfortnite)

(https://web.archive.org/web/20180907123127/https://www.instagram.com/fortnite/)

(https://web.archive.org/web/20180907123127/https://vk.com/fortnite)

Home (/web/20180907123127/https://www.epicgames.com/fortnite/home)
Battle Pass (/web/20180907123127/https://www.epicgames.com/fortnite/battle-pass)
Watch (/web/20180907123127/https://www.epicgames.com/fortnite/watch-fortnite)
Get Fortnite (/web/20180907123127/https://www.epicgames.com/fortnite/buy-now/battle-royale)

News (/web/20180907123127/https://www.epicgames.com/fortnite/news)
FAQ (/web/20180907123127/https://www.epicgames.com/fortnite/faq)
EULA (/web/20180907123127/https://www.epicgames.com/fortnite/eula)
Help (https://web.archive.org/web/20180907123127/http://fortnitehelp.epicgames.com/)

US/CANADA